A Testbed for Hardware-assisted Online Profiling of IoT devices*

Shangrong Li
School of Information Engineering
Chang'an University
Xi'an, China
shangrong.li@foxmail.com

Junyan Ma †
School of Information Engineering
Chang'an University
Xi'an, China
alexmajy@acm.org

Yi Li
School of Information Engineering
Chang'an University
Xi'an, China
seven.yi.lee@gmail.com

ABSTRACT

The widespread application of the Internet of Things (IoT) has put forward higher requirements for the reliability of the IoT devices. Traditional testing methods, while able to get a rough approximation of the performance of IoT devices, often fail to extract detailed runtime execution traces of applications from the resource-constrained devices. Hardware-assisted tracing can make it easier for IoT developers to obtain the rich running information of IoT devices with limited overhead, which brings new possibility to fully evaluate the software (i.e. firmware) of the IoT. For the hardware-assisted tracing data, existing offline analysis methods have severely limited the observable time span due to a large amount of tracing data. In this paper, we propose an FPGA-based online profiling testbed to carry out real-time processing of tracing data and implement continuous observation of IoT devices. Our preliminary experiment shows that the testbed has superior performance in terms of runtime trace capturing and sampling frequency. Finally, the potential applications of the captured traces are discussed.

CCS CONCEPTS

• Computer systems organization → Embedded and cyber-physical systems

KEYWORDS

Internet of Things, FPGA, Online profiling, testbed

ACM Reference format:

Shangrong Li, Junyan Ma and Yi Li. 2020. A Testbed for Hardware-assisted Online Profiling of IoT devices. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops(ICSEW'20). ACM, New York, NY, USA, 4 pages.* https://doi.org/10.1145/3387940.3392247

* The work is supported by National Key R&D Program of China 2018YFB1600800.
† Corresponding author

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea
 978-1-4503-7963-2/20/05 $15.00
https://doi.org/10.1145/3387940.3392247

1 Introduction

IoT has become a valuable tool for observing the natural world. Therefore, in the natural environment, the hardware and software components of the IoT system need to be highly reliable and well tested. Traditional testbeds, while able to obtain an approximation of the performance of IoT devices, often fail to capture detailed operational information. At present, mid-to-high-end MCUs (Microcontroller Unit) usually integrate dedicated debugging modules, which can be used for tracing and analysis of program execution. For example, there are three tracing modules in the ARM Cortex-M3 core. ITM (Instrumentation Trace Macrocell) & DWT (Data Watchpoint and Trace) supports interrupt event tracing, PC sampling and various performance counters. The ETM (Embedded Trace Macrocell) can trace every path the MCU executes. Hardware-assisted tracing technology can trace the execution process and events of the MCU without affecting performance. It can mark some specific locations where the program is running and record the program running trace. For Internet of Things, applications usually involve interaction of multiple nodes for long-term time span. Therefore, a large amount of raw tracing data will be generated, which limits the observable time span of offline analysis methods. Meanwhile, it takes a lot of time for developers to analyze these the data offline, which makes the entire testing process inefficient. In this paper, we propose an FPGA-based testbed to process tracing data online. This method supports arbitrary observation time and can increase the capture of information, which effectively solves the problem of limited test information collection and processing capacity of various testbeds.

Related Work. With the development of software testing technology, most of the testing methods and tools can be directly or indirectly used in embedded software testing. However, due to the particularity of embedded software, such as hardware coupling, real-time, inconsistent development, and running environment, insufficient resources, numerous CPU types and so on, the testing method of embedded software is very different from that of traditional commercial software. Currently, there are two methods for embedded software testing: software instrumentation and embedded debugging interfaces.

Software instrumentation is easy to use and well supported by tools. Although it can capture some valid test information, the test

code has increased redundancy and requires additional resource overhead. For example, Flocklab [1] uses software instrumentation to mark software execution paths, and uses GPIO (General Purpose Input Output) interfaces to trace and record time-sensitive events of target nodes.

The method of using the embedded debugging interface provided by the MCU to obtain embedded software test information has the advantages of low intrusion, small impact on the software under test, and strong real-time performance. AVEKSHA [2] uses a custom debug board to connect with OCDM (On-Chip Debug Module). This debug board supports 8 types of triggers and statistical analysis of applications using PC (Program Counter) polling. It is costly and has a large runtime overhead. Minerva [3] uses the JTAG (Joint Test Action Group) port to perform memory tracing and global assertion, but the JTAG debugging method cannot meet the debugging needs because it frequently switches between suspend and resume. The existing hardware-assisted tracing testbed HATBED [4] can provide network-level remote debugging, flexible software tracing, and non-intrusive software analysis and realize non-intrusive tracing and analysis of embedded systems without relying on operating systems and applications. Based on FPGA, Decker [5] proposed a scheme for online analysis of multi-core processor test information. This scheme can process the trace data stream in real-time, to continuously observe the state of the SoC (System on Chip). However, this method of test information flow processing has not been applied to the testing of IoT systems.

Research Motivation. HATBED observers use the lower-performance CY7C68013A logic analyzer and Raspberry Pi device to complete the capture and processing of test information. HATBED currently has the following shortcomings:

1. The sampling frequency of the CY7C68013A logic analyzer is up to 24MHz. The maximum SWO (Serial Wire Output) output frequency of the target that can be captured is 8M (3 times the sampling frequency), and it is not possible to capture the test information output by the target at a higher frequency.
2. Due to the bus polling mechanism of the Raspberry Pi 3B+, the packet loss rate of data reception is high, and the long-term capture of test information cannot be achieved.

To solve the existing shortcomings of HATBED, we designed and proposed an online analysis testbed based on the FPGA. On the one hand, our testbed has greatly improved in terms of runtime trace capturing and sampling frequency. On the other hand, our testbed has the potential benefits for testing and profiling of software on IoT devices.

2 Testbed System Architecture

The testbed consists of targets, observers, and controller. Targets are the nodes of the IoT systems. Distributed observers monitor these nodes, and the observers perform clock synchronization through the NTP protocol. The controller is a high-performance computer server that can control the entire testbed. Figure 1 shows the structure of the testbed.

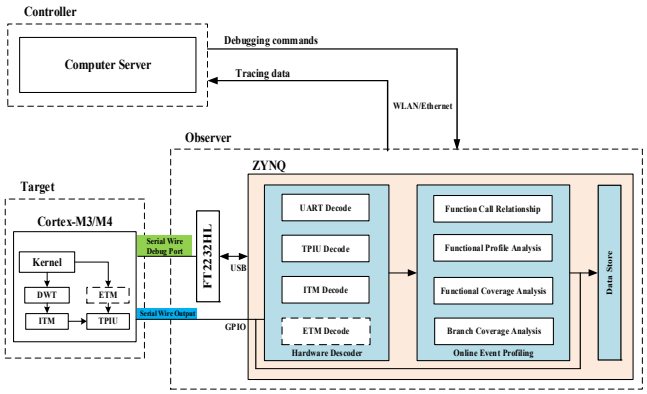

Figure 1: The testbed architecture

2.1 Testbed Target

The MCUs of target nodes have ARM Cortex M3 / M4 cores. Not all MCUs equipped with this core have ETM units, but almost all of them have ITM units, DWT units, SWD (Serial Wire Debug) interfaces and SWO interfaces. Through these debugging units and interfaces, test information can be output.

2.2 Testbed Observer

The observer consists of a development board containing Xilinx Zynq-7000 SoC chip (Xilinx Zynq XC7Z020) and FT2232HL debug board. The development board and the target node are connected through the FT2232HL debug board. The development board captures the test information output by the target node through the GPIO port. The development board is equipped with 1G DDR, 8G EMMC, 32G TF card, 2 Gigabit Ethernet interfaces, 4 USB interfaces and other peripherals for Zynq-7000 SoC chip. The PS (Processor System) part of Zynq-7000 SoC runs the ubuntu16.04 operating system and runs OpenOCD (Open On-Chip Debugger) and GDB (GNU Project debugger) on the operating system to debug the tested target. Zynq-7000 SoC can provide test information hardware decoding, online event profiling, data storage, and clock synchronization functions:

Hardware decoding. The test information decoding algorithm rewritten by the hardware description language is run through the SoC PL (Programmable Logic, i.e. FPGA) part. The debugging data stream captured by the GPIO port can be decoded by UART (Universal Asynchronous Receiver/Transmitter), TPIU (Trace Port Interface Unite), ITM, and ETM. On the one hand, it can reduce the storage amount of test information. On the other hand, it also speeds up test information processing.

Online event profiling. The functions include function call relationship analysis, function profile analysis, function coverage analysis, branch coverage analysis and et al. Through the online profiling of the decoded results, a basic analysis report of the test program is obtained for users to view.

Data storage. The Zynq SoC supports saving the original test information and analysis report to the local through the file system, and then uploading it to the testbed controller via Ethernet.

Clock synchronization. The clocks are synchronized between the observers using the method of HATBED. Select an observer as the clock source and obtain accurate clock with GPS satellite clock data and PPS signals. All observers are clock aligned with this clock source via the NTP protocol.

2.3 Testbed Controller

A high-performance computer server acts as a controller to control the entire testbed. The controller adds the tracing configuration code to the project under test and distributes it to the observers after compilation. The controller controls the observers through scripts to stop, halt, resume, polling memory, and receive tracing data. The controller can present the analysis report to the user for viewing. The controller can also run Sigrok software, use TPIU, DWT, ITM and ETM decoders to complete the original data decoding, and extract the relevant tracing information.

3 Benchmarks

The benchmarks focus on the performance of the testbed in trace capturing and sampling frequency. The experiment uses STM32F103VET6 (Cortex-M3 core) development board as the target node. The bare-metal based standard mailbox queue project code is used for testing, and 2 DWT watchpoints are enabled in the program. To reduce code uncertainty, the standard code is generated by STM32CubeMX software.

3.1 Collection Time Evaluation

IoT systems run programs such as sensor data collection and processing, real-time operating systems and node communication. This results in the embedded system taking a long time to run a program. Therefore, to comprehensively analyze the program execution trajectory, the testbed should have the ability to capture test information for a long time.

Table 1: Mean Statistics of Zynq and Raspberry Pi Sampling Points

Group	1	2	3
Raspberry Pi	185037	107008	147129
Zynq	24000000	24000000	24000000

For SWO at 8MHz output frequency, Raspberry Pi and Zynq respectively used 24M collection frequency to collect test information. In the actual test, the collection time was set as 1s. Three groups were counted in the experiment, and 10 times of sampling points were collected for each group. The mean of sampling points of each group was calculated as shown in Table 1. The number of sample points obtained with Raspberry Pi is less than 24,000,000. Through Zynq collection, 24,000,000 sampling points can be obtained for each collection, without loss of sampling data.

Many devices of Raspberry Pi share the same bus with the USB interface. During the collection process, if other devices transmit data through the shared bus, the collection process will be terminated. The data collected using Zynq is buffered by the DDR3 device and stored locally. This method can ensure continuous data collection for several minutes. Therefore, the testbed can analyze the program execution trajectory for a long time.

3.2 Sampling Frequency Evaluation

In addition to the amount of DWT set in the program, the test information output of the target node is also related to the output frequency of SWO. To verify the necessity of higher sampling frequency in the capture of test information, the logic analyzer CY7C68013A and Zynq were used to capture the SWO output information at different sampling frequencies.

Table 2: Statistics of decoded events at different output frequencies of SWO

SWO	Device	DWT	Exception	PC	Overflow
8MHz	CY7C68013A	54	77	1756	75
12MHz	Zynq	55	112	1756	61
18MHz	Zynq	56	132	1756	43
24MHz	Zynq	57	142	1756	32
36MHz	Zynq	58	170	1756	10

As shown in the Table 2, since the maximum sampling frequency of logic analyzer CY7C68013A and Zynq is 24MHz and 250MHz, respectively. The logic analyzer is used to capture the information of SWO at 8MHz output frequency. Zynq is used to capture the information of SWO at 12M, 18M, 24M, and 36M output frequency. The sampling frequency is 3 times of SWO output frequency. After the testbed controller decoded the test information through Sigrok software, we counted the number of DWT, Exception trace, PC sampling and overflow events captured by the program during one run cycle. Overflow events occur because the amount of output per unit of time exceeds the SWO port bandwidth, which usually results in the loss of some trace data.

Analyzing the experimental results, it can be concluded that the number of Overflows gradually decreases with the increase of the SWO output frequency. Increasing the SWO output frequency from 8M to 36M can reduce the number of overflow events by 86.6%. Under different SWO output frequencies, Overflow events will not affect the capture of PC sampling events but will lead to the loss of DWT events and exception trace events, so that the functions executed and exceptions generated during the running process of the program cannot be comprehensively captured.

The testbed uses GlobalCounter to encode the functions to be observed. DWT Watchpoint monitors GlobalCounter and records the PC and the encoded value when it is assigned. The GlobalCounter codes of the MailProducerTask function and the bubble_sort function is 0x0A and 0x0D, respectively. As shown in Figure 2, the generated information exceeds the SWO output bandwidth when the SWO output frequency is 8 MHz. Therefore, overflow causes DWT events that represent the MailProducerTask

function to be lost and only output the DWT event representing bubble_sort function. When the output frequency of SWO is increased to 36MHz, all encoding functions are captured, and DWT events that represent the task function and bubble_sort function can be output in sequence according to the trigger order.

Therefore, by increasing the output frequency of the debug trace port, more comprehensive and effective trace data can be obtained. Zynq, as an observer of the testbed, is fully capable of capturing test information by SWO at higher output frequencies.

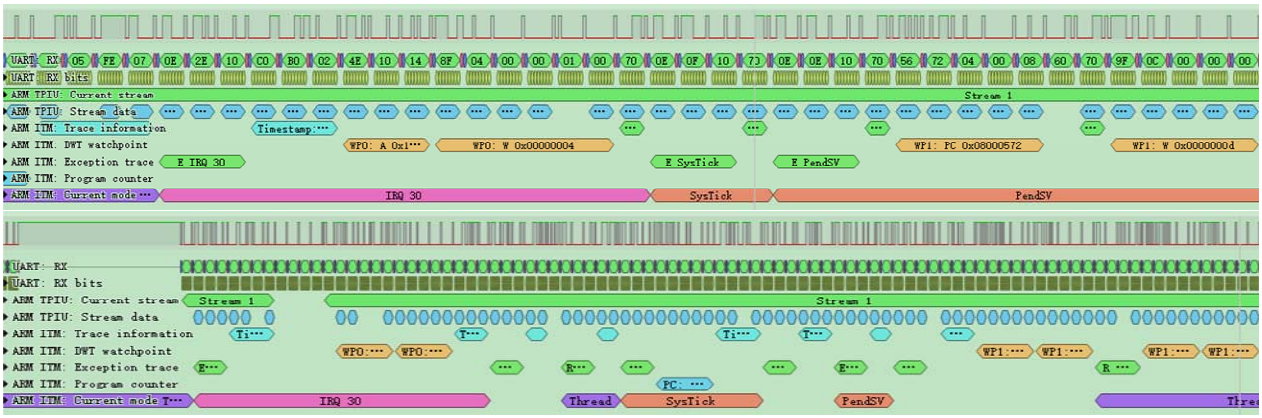

Figure 2: The results of test information decoding of SWO at the output frequency of 8MHz (Above) and 36 MHz (Below) respectively

4 Applications

Coverage criteria, such as function coverage, statement coverage and basic block coverage, are the most common metrics used for evaluation of software quality. The runtime execution traces collected through the testbed, including PC samples and events of interest encoded by DWT Watchpoint, can be used for measurements of the metrics by leveraging source code and the information extracted from executable files. Moreover, the measurements can be optimized and synthesized into hardware circuits on FPGA for online profiling of IoT devices, which is the next focus of the testbed.

To verified the feasibility of the testbed in the real-world, we measure the function and basic block coverage of the UDP communication process of Generic (GNRC) network stack in RIOT[1] which is an open-source operating system for IoT Devices. The target nodes are two STM32F103RCT6 development boards equipped with CC1101 wireless modules. During the test, the two nodes are configured as UDP servers and send UDP messages to each other. The entire process is recorded by the testbed (without ETM, DWT Watchpoint is used to detect the relevant function entry point), and the recorded tracing data is used for functions Coverage and basic block coverage analysis.

Function coverage. There are 18 functions related to the UDP communication process. We use a global counter to encode 8 of them and instrument at the function entrances. The traces of PC samples and encoded counters are used for function coverage. Finally, 16 of the 18 functions are captured and the function coverage rate is 88.9%.

Basic block coverage. We use PC samples to measure basic blocks coverage of upd_cmd function in UDP related code, and the basic blocks of functions called in upd_cmd is considered during the measurement. According the PC samples in the traces, 19 of the 55 basic blocks is executed and the basic block coverage is 34.5%.

5 Conclusion

The paper proposes an FPGA based testbed for the IoT devices with modern 32-bit MCU architecture. The benchmark tests verified that the performance of the testbed compared to HATBED has greatly improved in terms of runtime trace capturing and sampling frequency, which can enable comprehensive analysis of runtime traces for IoT devices. The discussion of applications of the captured traces shows that our testbed has the potential benefits for testing and profiling of software on IoT devices.

REFERENCES

[1] Lim, Roman, et al. "Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems." *Proceedings of the 12th international conference on Information processing in sensor networks*. 2013.

[2] Tancreti, Matthew, et al. "Aveksha: A hardware-software approach for non-intrusive tracing and profiling of wireless embedded systems." *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. 2011.

[3] Sommer, Philipp, and Branislav Kusy. "Minerva: Distributed tracing and debugging in wireless sensor networks." *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. 2013.

[4] Yi, Li, Junyan Ma, and Te Zhang. "HATBED: a distributed hardware assisted testbed for non-invasive profiling of IoT devices." *Proceedings of the 2nd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things*. 2019.

[5] Decker, Normann, et al. "Online analysis of debug trace data for embedded systems." *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018.

[1] https://riot-os.org/